2009-08-01

# Reducing Source Load in BitTorrent

Brian Sanderson
bts7@byu.net

Daniel Zappala
daniel_zappala@byu.edu



## Original Publication Citation

Brian Sanderson and Daniel Zappala, "Reducing Source Load in BitTorrent", The 18th International Conference on Computer Communications and Networks (ICCCN 29), August 29.

# Reducing Source Load in BitTorrent

Brian Sanderson and Daniel Zappala
Computer Science Department, Brigham Young University
bsanderson@byu.net, zappala@cs.byu.edu

*Abstract*— **One of the main goals of BitTorrent is to reduce load on web servers by encouraging clients to share content between themselves. However, BitTorrent's current design relies heavily on the original source to serve a disproportionate amount of the file. We modify standard BitTorrent software so that a source determines the current popularity of each of the blocks of a file and tries to serve only those blocks that are rare. Using extensive PlanetLab experiments, we show that this modification can save a significant amount of the source's upload bandwidth, with the tradeoff of some increased peer download time. In addition, there are individual experiments that both save bandwidth and have a faster download time than standard BitTorrent. We examine some of the more exceptional experiments, explore alternative algorithms, and provide insight for further improvements.**

## I. Introduction

BitTorrent has become a very popular peer-to-peer application, accounting for as much as 80% of the traffic on backbone Internet links [1]. One of the keys to this popularity is that BitTorrent allows an organization without a lot of bandwidth to deliver files to a very large audience. BitTorrent clients share the content they download, so that some of the load that would be directed toward a centralized server is instead spread to the users of the system.

An important part of the BitTorrent system is its use of seeds to help distribute a file. When an organization wishes to share a file using BitTorrent, it runs a tracker, which helps peers locate each other, and an initial seed, which has the entire file available. Clients contact the tracker to find peers that are in the system, then contact these peers to find out which blocks they have. Peers download blocks in parallel from both the original seed and other peers. Once a peer has downloaded the entire file, it can itself become a seed.

One of the problems with BitTorrent, however, is that the initial seed, which we call the source, often uploads the majority of the blocks in the system. BitTorrent provides incentives that encourage peers to share in order to download from each other, but free-riders can download from the source without having to share any blocks [2]. To alleviate the high load on a server, most BitTorrent software allows the user to set a bandwidth limit, which caps the upload rate for the source and the seeds. However, this kind of limit is inflexible for an organization. Most sites don't mind helping a client to download a file quickly when there is no alternative, but would like to limit their upload rate if the file is easily available elsewhere.

In this paper, we modify the BitTorrent software to help reduce source load. The key idea of this modification is that the source should only upload blocks of a file that are rare. If a block is held by a significant number of peers, then clients should get that block from their peers, rather than from the source. The source then exists primarily to help clients who can't otherwise find a block and ensures that the file remains available in the system. This modification helps the source to use significantly less than the configured rate cap when other peers are available to share the load, and also reduces the effectiveness of free-riding.

Our BitTorrent modification, which we call the *scout*, includes two new mechanisms. First, the scout estimates block diversity, which is the popularity of each of the blocks of the file, using information from its directly-connected neighbors and by probing additional peers. Second, the scout uses an algorithm to decide which blocks to advertise as available to its neighbors. Advertising the same rare blocks to all peers may not be the best strategy, since giving different blocks to different peers will increase the chance that peers can get blocks from each other.

To determine the effectiveness of the scout, we run a series of experiments on PlanetLab, isolating the performance of the source from that of other seeds. Our results vary significantly, due to the unpredictability of available bandwidth on a global testbed. Averaged over all experiments, with various algorithms and settings, our BitTorrent scout saves about 50% of the source's upload bandwidth but extends the peer download time by 2.5 to 3 times. There are individual experiments that make better tradeoffs, and some both save bandwidth and have a faster download time, indicating there is likely room for improvements. These are encouraging results, considering the fact that in our experiment peers leave the system right after they download the file. In more typical scenarios, peers will act as seeds and help to maintain good download times even when the source restricts its uploading.

To provide further insight into what works best, we examine different advertisement algorithms and find that they should include some degree of randomness to prevent peer starvation. We also find that the source's neighbors provide enough information to form a good estimate of block diversity, even when the torrent has about 400 active peers. Overall, our results indicate that further study may lead to ways to provide better performance tradeoffs.

## II. Motivation

In a traditional client-server download, the server can quickly expend all its upload bandwidth when it must serve many clients. With BitTorrent, there is greater opportunity to distribute the load of providing a file among the clients in

978-1-4244-4581-3/09/$25.00 ©2009 IEEE

| Experiment | Source | Peer | Ratio |
|---|---|---|---|
| No Rate Limit | 2979 KB/s | 366 KB/s | 8/1 |
| 75 KB/s Rate Limit | 52 KB/s | 26 KB/s | 2/1 |

TABLE I

SOURCE VERSUS PEER UPLOAD RATE

the system. However, the tendency of BitTorrent is to heavily utilize the upload bandwidth of the source, with the peers playing a secondary role. Ideally, the source should upload little to nothing if block diversity is high and shouldn't upload significantly more than the peers in the system.

To illustrate the unfairness that can occur with BitTorrent, we conducted several experiments on PlanetLab. We created a torrent for a 200 MB file, hosted on a PlanetLab machine with a high-speed Internet connection. We then had as many PlanetLab hosts as possible join this torrent over a 30 minute period, typically getting about 300 to 400 hosts participating. Hosts take anywhere from 2 to 24 hours to download the file. In the first set of experiments, we did not configure BitTorrent with a rate limit, allowing peers to download from the source or peers as fast as they could, and peers lingered for 5 minutes after they finished their download. In the second set of experiments we set a rate limit of 75 KB/s on both the source and the peers, and the peers left the system immediately after finishing the download.

Based on these experiments, it is clear that the source often uploads far more than the peers. Table I shows that the source upload rate is 8 times higher than the peers without rate limiting, and twice as high with rate limiting. This occurs because a BitTorrent source serves blocks to the peers who have demonstrated they have the highest download rates. This policy is intended to spread the distribution of the file as quickly as possible, so that these fast peers can share the file with others. However, it also leads to the source using nearly as much bandwidth as it is allowed. Our goal is to reduce the amount the source uploads by not distributing content that is readily available elsewhere.

## III. BITTORRENT SCOUT

We implement a BitTorrent scout that reduces source load by only serving blocks that it considers to be rare. The goal of this system is to encourage clients to download blocks from other peers whenever possible, so that the source can save its bandwidth for those blocks that few peers have available.

To perform this function, the scout tries to estimate the diversity of each block, which is defined as the number of peers holding a block divided by the total number of active peers. A block diversity of 1.0 means that all active peers hold that block. Because both the numbers of active peers and the blocks they hold may vary rapidly, the scout samples the peers to get an estimate of block diversity.

To control which blocks a peer may download, the source determines which blocks it will advertise at the start of a connection with a peer. In BitTorrent, each time two peers connect to each other, they exchange a bitmap that lists the blocks they hold. Typically, a source advertises that it has all of the blocks of the file. Our scout simply modifies this advertisement so that it only includes those blocks that it considers to be rare. Later, a source may send an update with additional advertised blocks, but it cannot take back an advertisement once it has been made.

### A. Estimating Block Diversity

In the standard BitTorrent implementation, any peer can connect to the source at any time. We call the current set of connected peers the source's *neighbors*. Based on the blocks that neighbors advertise to the source, the scout can form an initial estimate of the block diversity. It then improves this estimate by periodically connecting to additional peers, listening to their advertisement (which is less than 100 bytes), and then disconnecting.

The scout must balance its desire to have a more accurate estimate of block diversity with the amount of bandwidth it consumes. The more frequently the scout polls additional peers, the more accurately it can estimate block diversity. However, if the scout probes neighbors too frequently, it can consume a significant portion of bandwidth, which would defeat the purpose of the solution. Accordingly, the scout is limited to contacting $r$ peers per minute. Our experiments compare different values of $r$ and their effectiveness at estimating the actual block diversity, as well as the saved bandwidth as compared to the original source implementation.

To track block diversity, the scout uses an indexed queue to store peer advertisements that it has collected. Each entry in the queue contains the peer identifier for the peer contacted, the list of blocks the peer advertised as available, and the timestamp of the last time the peer was contacted. The queue is sorted by this timestamp, so that the oldest information is at the front of the queue. Entries are expired from the head of the queue when their timestamp is 20 minutes old.

The scout adds entries to the advertisement queue in one of two ways. First, any time a neighbor sends an advertisement, or updates its advertisement, the scout adds or updates the entry for that peer. Second, the scout probes $r$ new peers per minute, using a list obtained by contacting the tracker. In both cases, the peer may already have an entry in the queue, so we update that entry as needed. If a peer becomes unreachable, we remove its information from the queue.

Using the advertisement queue, the scout estimates the diversity of each block $i$ as $c_i/n$, where $c_i$ is equal to the number of peers in the queue that have that block and $n$ is the total number of peers in the queue.

### B. Advertising Blocks

Whenever a new neighbor connects to the scout, it chooses a set of blocks to advertise to that neighbor. We implement the following selection strategies:

- *less than $k$:* The source selects every block $i$ where $c_i/n < k$. In our experiments, we use $k = 0.4$, which means that the source will advertise the rarest 40% of blocks to peers.

- *probabilistic:* The source chooses a uniform random number in the range 0 to 1 for each block, $i$, and advertises that block if this number is greater than $c_i/n$. This increases the chance that different neighbors are given different sets of blocks.
- *shuffle rare:* The source creates a unique ordering of the blocks for each neighbor, based on the peer's identifier and IP address. For each neighbor, the source selects the first $2z$ blocks in this set, where $z$ is the number of blocks held by less than 40% of the peers. The source then advertises the rarest $z$ blocks in the set. This algorithm is also designed to give different neighbors different sets of blocks.
- *random:* The source chooses a random set of $z$ blocks to advertise. This strategy is used as a comparison, to verify that more intelligent strategies do indeed have an advantage.

To prevent starvation in all algorithms, we periodically repeat this process and advertise any newly-selected blocks in an update message to the current neighbors.

## IV. Methodology

We use PlanetLab to run experiments with the BitTorrent scout and compare its performance to a standard BitTorrent implementation. We chose to run a measurement study rather than a simulation of BitTorrent, so that we can have higher confidence that our enhancements will work in practice, with official BitTorrent source code and live TCP streams. While PlanetLab hosts do not perfectly model home users, they do have geographical diversity and different service providers, traffic policies, and available bandwidth; PlanetLab is currently our best estimate of "reality" when running a controlled experiment. A downside of using PlanetLab for a measurement study is that it introduces a high amount of variance, due to the heterogeneity of the hosts and networks we use. This makes it difficult to obtain repeatable and consistent results.

For the set experiments we report here, we use BitTorrent 4.0 [3] to distribute a 200 MB file, hosted on a server at BYU with a 100 Mbps Internet connection, with no peer-to-peer throttling by the BYU routers. Distributing a file much larger than this exceeds PlanetLab's byte limits, resulting in rate-limiting that would invalidate our experiments. We configure BitTorrent to limit the upload rate of all peers to 75 KB/s, to avoid disruptions to the hosting sites. Some peers upload at much lower rates, based on connectivity or site limits.

Our experiments are designed to gauge how effective the scout can be at the start of a torrent, when load is highest. Accordingly, we configure all peers to leave the system as soon as they have downloaded the file. This prevents any peers from becoming seeds, creating a worst-case scenario for load on the source. The performance of the scout will thus likely be better than what we report here, since seeds will provide extra capacity and offload some of the burden from the source.

To measure the performance of the BitTorrent software and our modifications, we modify BitTorrent to record all messages exchanged between peers in the swarm, as well as pertinent events. To correlate events across all of the peers, we use SNTP to determine the offset of the local clock relative to an NTP server, then factor this offset into the timestamp in each message log. This method allows us to align logs on different peers with minute-level accuracy, despite significant clock drift on PlanetLab, and is sufficient for our purposes.

To run an experiment, we have as many PlanetLab hosts as possible join the torrent over a 30 minute period, typically getting about 300 to 400 hosts participating. Hosts take anywhere from 2 to 24 hours to download the file. We terminate the experiment once 95% of the hosts have downloaded the file, since some hosts experience persistent connectivity problems and others transfer at rates less than 2 KB/s. All together, the time to prepare, run and collect data for an experiment takes about two full days. We ran a total of 63 experiments over a one year period, logging 350 GB of messages and consuming 150 TB of BYU's upload bandwidth.

The results we present here are taken from 63 experiments, varying the advertisement algorithm and $r$, the number of peers probed per minute. We vary $r$ from 0 to 20, in increments of 5, and run one experiment for each value with each advertisement algorithm. We then repeat the most promising experiments up to 8 times. We also run 5 experiments with the standard BitTorrent implementation. To avoid biasing one variation over another, we randomize the times they run, so that each variation has an equal chance at running in different weeks during the year.

## V. Results

Our experiments vary greatly from run to run, even among experiments with identical settings. The main reason for this is that network configurations, background traffic, and competing PlanetLab experiments all change from day to day and month to month. There is also variability with any BitTorrent download, since performance depends on which peers a client connects to, the connection quality between peers, the blocks the peers have available, and whether those peers allow the client to download from them. The scout introduces additional variability since it decides which blocks the source advertises to each peer.

When using the scout, there is a general trade-off between upload amount and download time. Figure 1 plots the results of each experiment configuration, showing the amount of data uploaded by the source over the entire experiment and the average peer download time. Each point on the graph represents a single 24-hour experiment – the color of a point represents the algorithm and the shape represents the value of $r$. The standard BitTorrent implementation is shown in green.

It is immediately apparent that the scout enhancements do reduce source load. There are many experiments in which the source uploads less than 1 GB total, meaning it uploads the file under 5 times to serve about 400 nodes. In general the more the source reduces the amount of uploaded bytes, the longer peers take to download the file. The penalty for cutting the source's uploaded bytes in half is that we increase the average peer upload time by about 2.5 to 3 times. Given this

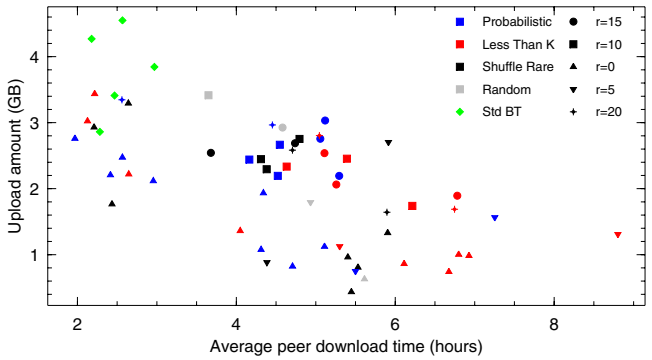



Fig. 1. Overall experiment results

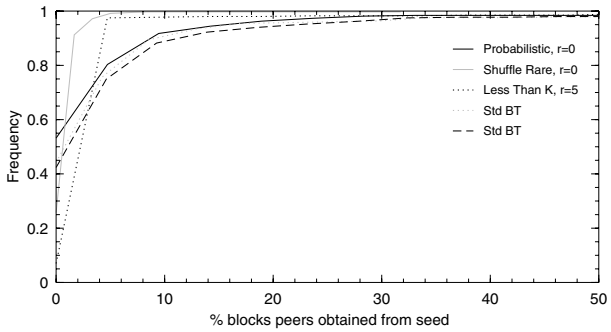



Fig. 2. Interesting experiments: CDF of source dependence

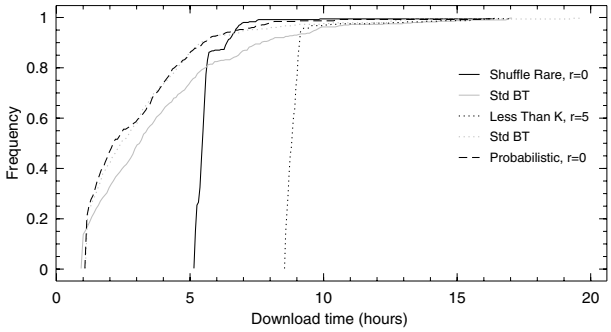



Fig. 3. Interesting experiments: CDF of peer download time

trade-off, it is difficult to say which point on the spectrum is best. To a content-distributor the upload amount may be most importance, while to the peers a small download time is the most important. The slope of the trade-off line will likely depend on peer lingering time; in these results, peers do not linger at all once they finish the download. We plan additional experiments to examine this effect.

There are some notable exceptions to this tradeoff. Experiments with $r = 0$, meaning the scout did not probe additional peers, often do better than standard BitTorrent, both in terms of upload amount as well as download time. These include the *probabilistic* (blue upward triangles), *less than K* (red upward triangles), and *shuffle rare* (black upward triangles), all on the left-hand side of the graph. The scout, in this case, is still advertising only rare blocks, but is computing block diversity using only the information from its directly connected neighbors.

There are several reasons why no additional probing is needed for good performance in our experiments. First, a BitTorrent client connects to every peer it is given by the tracker, typically about 80 peers. As a result, many peers end up being directly connected to the source. Although only a few of these download from the source at the same time, each connection sends messages to the source informing it when blocks have been obtained. In our experiments, as many as 100 peers have been directly connected to the source at one time, meaning the source has complete and up-to-date knowledge of the block diversity for 25% of the peers in the system. A probe from the scout is more uncertain than a direct connection, because with a probe the scout only obtains a snapshot of the peer's available blocks. Thus with very complete information from neighbors, probing may actually worsen the scout's estimate of block diversity. It is possible that probing may only help when there are very large numbers of active peers.

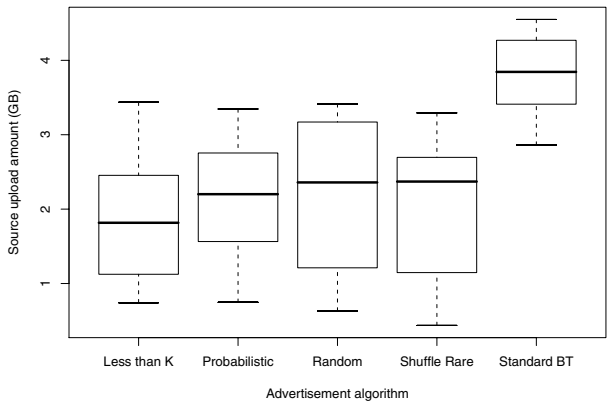
### A. Understanding the Exceptions

We picked a few of the exceptional experiments to try to understand why they do not follow the general trend, performing either exceptionally well or exceptionally poorly. We chose five such experiments, which will be referenced in the following discussion. These include *probabilistic,* $r = 0$, the left-most experiment in Figure 1; *shuffle rare,* $r = 0$, the bottom-most experiment in the same figure; *less than K,* $r = 5$, the right-most experiment in the figure; and the bottom-most and right-most standard BitTorrent experiments.

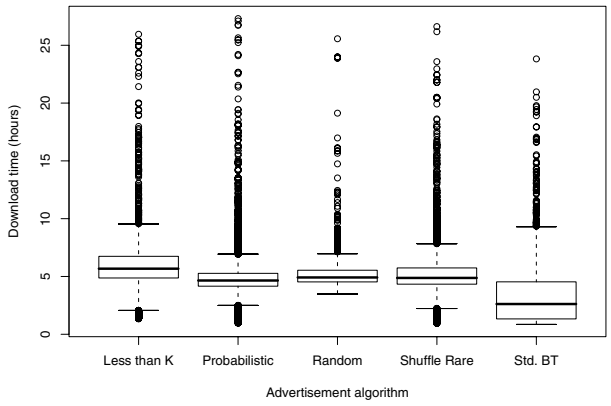
For these five experiments, Figure 2 shows a CDF of the percentage of blocks peers obtain from the source and Figure 3 shows the CDF of the peer download time. The *probabilistic* algorithm with $r = 0$ performs similar to the two standard BitTorrent experiments, with 80% of the peers downloading 2% to 4% fewer blocks from the source, without increasing download time. The other two experiments have significantly different performance.

With *shuffle rare* and $r = 0$, the source gives most peers fewer than 2% of the blocks, forcing them to get the rest from their peers. While the 80th percentile is about the same as the above experiments, nearly all peers take 5 hours to download the file, rather than allowing some peers to download more quickly. This approach appears to penalize the faster peers, making them download from the slower peers.

With the experiment using *less than K* and $r = 5$, the source gives out more blocks than *shuffle rare*, but peers take 8 to 9 hours to download the file. In this experiment, there were many peers who were in endgame mode for hours, waiting for one last block to be made available to them. The source had given this block to some peers, so it no longer considered it to be rare, but the peers that had this "valuable" block were either very slow or had connection problems with other peers. Once the slow peers with the valuable block finished and left the system, the scout discovered that this block was extremely

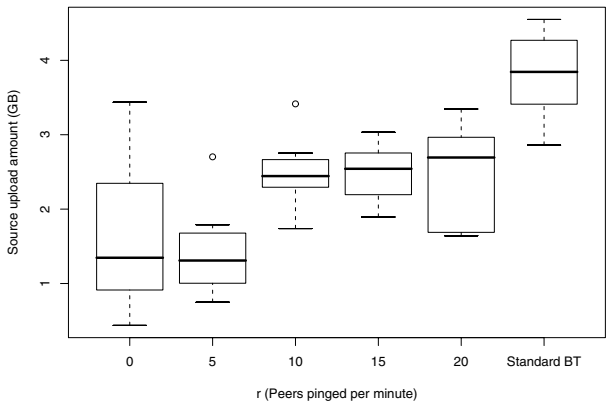



(a) Bytes uploaded by the source

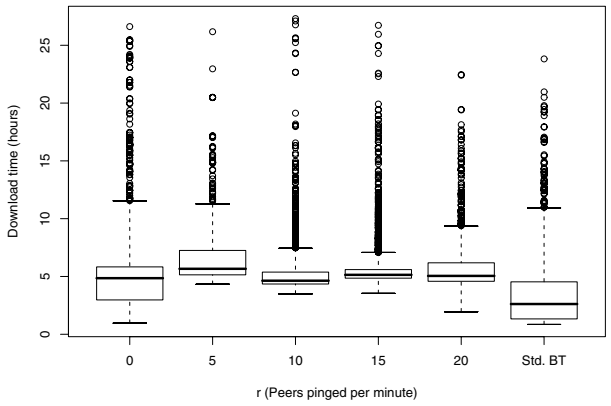



(b) Peer download time

Fig. 4. Algorithm Comparison




(a) Bytes uploaded by source




(b) Peer download time

Fig. 5. Probing Comparison

rare and gave it out, allowing the other peers to finish.

Returning to Figure 1, it is clear that *less than K* has the majority of the worst performances. This indicates how important it is to include some randomness in block advertisements. The *less than K* algorithm advertises a very similar block set to peers that connect in close intervals. The *probabilistic* and *shuffle rare* algorithms tend to avoid this problem.

### B. Comparing Algorithms and Probing Frequencies

To better compare the effect of different advertisement algorithms and probing frequencies, we combine the results of several experiments. To compare algorithms, we combine all the experiments for the same algorithm, regardless of $r$, and to compare probing frequencies we combine all the experiments for the same $r$, regardless of the algorithm. We then summarize the results using a box plot with the standard five-number summary. The box is drawn using the lower 25% quartile, median, and upper 75% quartile, and the whiskers represent the minimum and maximum of all data within 1.5 box-lengths from the 25% quartile and 75% quartile. Any data values that are outside the whiskers are outliers.

Figure 4 compares the algorithms based on the bytes uploaded by the source and the peer download time. All of the algorithms are able to decrease the median load on the source by about 40% to 50%, while the median upload time approximately doubles. The performance of all the algorithms is very similar, with a very tight bound on the average download time for the peers. Note that there appear to be a lot of outliers, but this is because there are about 4000 peers for each of the algorithms. The scout does not significantly increase the worst-case download time for peers.

Figure 5 compares different probing frequencies using the same metrics. The experiments with $r$ equal to 0 or 5 peers per minute result in much greater bandwidth savings – about 75%. In terms of download time, the experiments with $r$ equal to 0 have a 25% quartile that is much lower than when probing takes place. This means that 25% of all peers that use $r = 0$ finish in the same average time as downloads that used standard BitTorrent, while simultaneously saving 50% of the source's bandwidth.

Probing more peers actually increases the amount the source uploads, though it is still less than standard BitTorrent. While counter-intuitive, we believe this occurs because probing tends to contact peers only a few times during the entire experiment. The information from each probed peer is out of date relative to the source's neighbors, which send updates constantly, and these peers thus seem to have many fewer blocks than they really do. Thus the more probing that occurs, the source's estimate of block diversity becomes less accurate.

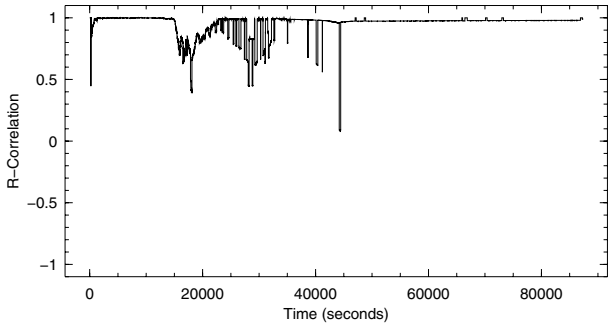



Fig. 6. Accuracy of block diversity estimate, *probabilistic*, $r = 0$

### C. Block Diversity Estimate

We next examine how accurately the source is able to estimate block diversity. We focus on the *probabilistic* algorithm with $r = 0$ to verify that directly connected neighbors are enough to produce an accurate estimate in our experiments. To correlate the data across peers, we align the logs based on their timestamps, which were roughly synchronized with SNTP. We then use the detailed peer trace to compute the R-correlation between the estimate and the actual diversity over time. R-Correlation values range from 1.0 to -1.0, where 1.0 indicates exact correlation, 0.0 indicates no correlation, and -1.0 indicates an inverted correlation between two data sets.

Figure 6 shows a strong correlation between the estimate and the actual block diversity for the *probabilistic* algorithm in this experiment. This shows that the directly connected neighbors provide enough data for a group of 400 peers.

## VI. Related Work

The super-seeding algorithm also tries to reduce the load on a BitTorrent source, but with a different method [4]. With super-seeding, the source acts as if it does not have any blocks, then as clients connect it tells them it has acquired one block. The source will not inform a peer of any other blocks until it can confirm that this block has been delivered to at least one other peer. Experiments on PlanetLab show this can save 20% of the source's bandwidth, with little effect on the overall time it takes for 90% of the peers to finish downloading the file. In this work, peers act as seeds after downloading the file, which provides a significantly different testing scenario than our work. If the source uploads less, there are plenty of other peers with the entire file that can help out, particularly once the experiment has been running for a while. As compared to super-seeding, the scout only shares blocks that are truly rare, and thus can reduce source load even further. Additional experiments are needed to determine whether the scout can effectively push load onto other seeds as with super-seeding.

Several other projects seek to improve BitTorrent in ways that are complementary to our work. Bindal et al. improves the efficiency of BitTorrent by modifying the tracker so that it gives clients a list of other peers that use the same ISP [5]. This ensures that most of the peer-to-peer traffic stays within the ISP, which saves outgoing bandwidth for that ISP. The smart seed algorithm has a seed decide which block to give out by examining the blocks needed by the peer and choosing the one the seed has served the least [6]. This results in the seed serving fewer redundant blocks, however this modification is not compatible with existing BitTorrent implementations. Finally, the Slurpie peer-to-peer system has several mechanisms that allow it to control the load on a central server regardless of the number of active peers [7].

## VII. Conclusion

This work shows that it is possible to reduce the source's upload bandwidth by modifying the source to selectively advertise blocks based on their popularity. Saving source bandwidth typically results in longer peer download times. However, in our best experiments the scout can reduce the source's upload bandwidth by 75%, while only doubling download time. This is an encouraging result, since none of the peers in our experiments act as seeds once they download the file. Having peers act as seeds, as is common with BitTorrent, will improve performance by providing additional upload capacity to offset the reductions by the scout. Our experiments also show that the advertisement algorithm should include some randomness, to avoid starvation, and that a source's neighbors provide a good estimate of block diversity for moderate numbers of peers.

There are a number of areas for future work. First, it is possible that we are being too aggressive in our attempts to reduce source load. We plan to focus on *probabilistic* and try different probability distributions to improve the performance tradeoff. Another way to improve download time is to treat peers differently when they are close to finishing. By giving a peer the final blocks it needs, the source should be able to avoid difficulties that arise when peers are waiting for a block that is only held by slow or poorly-connected peers.

We also want to address some of the limitations of our measurement study. We plan to run experiments where we add lingering time, so that there are additional seeds during the swarm, and to compare our performance directly to super-seeding. We are also planning to run simulations using the scout, so that we can examine performance in a more controlled environment and so that we can determine whether probing becomes valuable for much larger numbers of peers.

## References

[1] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot, "Packet-level traffic measurements from the Sprint IP backbone," *IEEE Network*, vol. 17, pp. 6–16, November 2003.

[2] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang, "Exploiting bittorrent for fun (but not profit)," in *IPTPS*, 2006.

[3] B. Cohen, "Bittorrent," http://www.bittorrent.com.

[4] Z. Chen, Y. Chen, C. Lin, V. Nivargi, and P. Cao, "Experimental analysis of super-seeding in bittorrent," in *IEEE Communications*, 2008.

[5] R. Bindal, P. Cao, W. Chan, J. Medval, G. Suwala, T. Bates, and A. Zhang, "Improving traffic locality in Bittorrent via biased neighbor selection," *ICDCS 2006*, February 2006.

[6] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and improving BitTorrent performance," Microsoft Research, Tech. Rep. MSR-TR-2005-03, March 2005.

[7] R. Sherwood and R. Braud, "Slurpie: A cooperative bulk data transfer protocol," in *INFOCOM*, 2004.